

# Summer of Code: Assisting Distance-Learning Students with Open-Ended Programming Tasks

## Conference or Workshop Item

# The Summer of Code: Assisting distance-learning students with open-ended programming tasks

Neil Smith
The Open University
Milton Keynes, UK
neil.smith@open.ac.uk

Mike Richards
The Open University
Milton Keynes, UK
mike.richards@open.ac.uk

Daniel G. Cabrero
The Open University
Milton Keynes, UK
daniel.cabrero@open.ac.uk

## ABSTRACT

A significant difficulty in teaching programming lies in the transition from novice to intermediate programmer, characterised by the assimilation and use of schemas of standard programming approaches. A significant factor assisting this transition is practice with tasks which develop this schema use. We describe the Summer of Code, a two-week activity for part-time, distance-learning students which gave them some additional programming practice. We analysed their submissions, forum postings, and results of a terminal survey. We found learners were keen to share and discuss their solutions and persevered with individual problems and the challenge overall. 93% respondents rated the activity 3 or better on a 5-point Likert scale (n=58). However, a quarter of participants, mainly those who described themselves as average or poor programmers, felt less confident in their abilities after the activity, though half of these students liked the activity overall. 54% of all participants said the greatest challenge was developing a general approach to the problems, such as selecting appropriate data structures. This is corroborated by forum comments, where students greatly appreciated "think aloud" presentations by faculty tackling the problems. These results strongly suggest that students would benefit from more open-ended practice, where they have to select and design their own solutions to a range of problems.

## CCS CONCEPTS

• **Social and professional topics** → **Computer science education**; *Adult education*; • **Applied computing** → Distance learning;

## KEYWORDS

Programming, Intermediate programmers, Distance learning

**ACM Reference Format:**
Neil Smith, Mike Richards, and Daniel G. Cabrero. 2018. The Summer of Code: Assisting distance-learning students with open-ended programming tasks. In *Proceedings of 23rd Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'18).* ACM, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.475/123_4


*ITiCSE'18, July 2018, Larnaca, Cyprus*

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

## 1 INTRODUCTION

Teaching programming has long been recognised as difficult [17] [5]. A key stumbling block in the development of expertise is moving from the rote application of basic programming constructs to the selection and application of appropriate algorithms and data structures to solve problems. This development of expertise can be influenced by many factors, but the amount of practice is significant [10] [7].

This paper describes the *Summer of Code*, a project was designed to give novice and intermediate programmers in a large distance-learning university an opportunity to practice and consolidate their programming skills.

A complicating factor for our students is that they are all studying part-time, learning at a distance, and typically at a study intensity of 30%–50% of full-time students. The low study intensity of part-time students means there can be long gaps between when they are introduced to a topic in one module and when they have to use and develop that skill in another. These gaps can lead to loss of ability, especially with novice programmers when the ability still depends on explicit, declarative knowledge in the student's mind [4].

Many of our students are mature students (only 8% of undergraduates are under age 21) [14], which means they need to juggling study with work and family commitments. These external commitments make it difficult for students to take even the minimal time required for study, let alone take on additional extra-curricular work; there is understandable reticence in the faculty to give the impression that students should take on additional work. The distance learning nature means there is a reticence to set students many practical programming tasks, due to problems of providing technical support. Students are remote from sources of support, which makes troubleshooting difficult. Distance-learning students also tend to struggle for long periods with minor problems before asking for help, when similar problems could be solved in moments by a TA walking through a lab.

The project had three main research questions:

(1) Will time-pressured part-time students seek out additional informal learning opportunities?
(2) Are mixed-ability communities of undergraduate students off-putting for students at either end of the ability scale?
(3) Do distance learning students exhibit similar stumbling blocks in their development of programs to solve problems?

We discuss related work in section 2. We describe the Summer of Code activity in section 3 and how we supported student participation. We analyse results from the tasks, student forums, and a terminal survey in section 4. Finally, we summarise what we have learnt in section 6 and describe our next steps.

## 2 RELATED WORK

The development of programming expertise has received much attention over many years, from Soloway's rainfall problem [12] to investigations into how students design software systems [6] [3]. The consensus is that students find programming a difficult skill to develop, especially the transition from novice to intermediate programmer. A key signifier of this transition is the student's understanding and use of "schemas" [17], which transfer knowledge from a declarative form to a more procedural form, reducing the cognitive load of programming [8]. As well as the explicit statement of schemas in teaching material, examination of worked examples and practice are key elements of developing programming skill [15] [7].

The distinction between declarative and procedural expertise in programming becomes important when we consider how expertise may degrade over time if it is not practised. Kim *et al.* [4] suggest that declarative knowledge deteriorates over time, sometimes catastrophically (leaving the learner unable to perform the task at all) while procedural knowledge is much more resilient. The longer gaps between programming activities for part-time students means there is more opportunity to forget key skills, making the reinforcement that comes from practice even more important.

Another factor in the design of the Summer of Code was the student population. An additional, voluntary, non-credit-bearing programme of study is more akin to a MOOC than a traditional module. With time-pressured part-time students, uptake and retention in the activity could be low. However, engaging materials and good interaction between students and instructors have been shown to greatly increase MOOC retention [2], and using tasks with at least the pretense of real-world relevance are more engaging than many seen in introductory programming courses [1] [9].

## 3 THE SUMMER OF CODE

The Summer of Code activity was developed to support students in the transition from novice to intermediate programming expertise. It was intended to both develop their skills and confidence in programming. The activity provided a low-stakes and supportive environment where students could practice and develop their skills.

The activity was not intended to *replace* any existing teaching, but to supplement and consolidate it; therefore, we decided to include only programing concepts which were covered in existing materials. The tasks had to be appropriate for the students' existing level of skill, and engaging to keep time-poor part-time students involved. Most tasks were solvable by students who had completed our introductory modules; two used additional techniques (graph search and dynamic programming) introduced in a second level algorithms module.

### 3.1 Activity structure

The activity comprised a series of ten programming puzzles over a two week period, with one puzzle released at midnight on each week day. Each puzzle was intended to require one to two hours of work to solve, including a number of problem analysis steps as well as programming. Students were encouraged to discuss the problems and share their solutions in a set of linked online forums; we provided support through these same forums. We provided an additional "day zero" puzzle before the main event started, to allow students to familiarise themselves with the style of puzzle.

Puzzles where presented to students using our existing online quiz platform. Each puzzle came in two parts, with the second part requiring some modification or extension of the first part. For each puzzle, the students were given the puzzle description (including a small problem instance) and a text file containing the data for the full task. The same puzzle input was used for both parts of the puzzle. The solution was typically a number or a few words of text.

Students implemented the solution on their own PC, using whatever language and development environment they wanted. We did not run any solutions on our platform. Students could submit their solutions as many times as they liked, with only their best result counting for completion. The puzzles did not require handling of malformed input and there were no trick questions.

Puzzle descriptions were presented in a light-hearted style, to reinforce the fun nature of the activity. Figure 1 gives an example; other puzzles involved processing instructions to drive a dot-matrix display board, word puzzles, and detection of interleaved subsequences in a set of strings. This format was based on the highly successful Advent of Code annual competition [16]. The full set of puzzles for the Summer of Code is available online [11].

**Figure 1: A sample Summer of Code task (worked example removed for brevity)**

After a fairly dull flight, you've finally arrived at your hotel. The good news is that the hotel has high-security electronic locks on the room doors. The bad news is that the staff are rather busy, and you think it will take a long time to get to your room.

Luckily, you know how their system works.

Each door in the hotel has a keyboard on the lock. You have to enter the correct two-letter code to get in to the room. Because the staff know that people won't remember the codes, they tell you a pass phrase you can use the generate the code from.

There's a long queue for people to be told how to generate the code from the pass phrase. You were here last year and you still remember how to do it.

You start with the first two letters of the pass phrase. This is the starting value of your code.

Then, for each subsequent letter in the pass phrase, you:

(1) "Add" the second letter of the code to the first letter of the code, replacing the first letter of the code
(2) "Add" the current letter of the pass phrase to the second letter of the code, replacing the second letter of the code
(3) Move on to the next letter of the pass phrase

"Adding" letters is done by converting the letters to their position in the alphabet (starting at one), adding, then converting the number back to a letter. For instance, to add `t` + `h`, convert them to numbers (`t`=20, `h`=8), add them (20 + 8 = 28), then convert back to a letter (28 is larger than 26, so it becomes 28 - 26 = 2, which is `b`).

All letters are converted to lower-case, and anything that isn't a letter is ignored.

For example, to find the code from the pass phrase `the cat`, the code starts as being the first two letters `th`, then the subsequent letters are used to update the code to give `vk`.

### 3.2 Student support

The puzzles were open to all undergraduate students. So that less skilled students would not be intimidated by the proficiency of more proficient students (some of whom were professional programmers, studying to deepen their skills), we created three forums: a normal "task discussion" forum, one called "No question too simple", and a "Show-offs' corner". The "no question too simple" forum contained guidance that all responses should be supportive and generous. The "show offs' corner" was described as being for activity that lay beyond the tasks themselves. Expecting that some students would be using the Summer of Code to explore new programming techniques, this forum was a place where they could discuss this activity without intimidating less proficient participants.

Faculty support for the activity was mainly participation in the forums. We also provided worked example solutions posted a few days after each puzzle was revealed, and video-conferenced interactive tutorials. Many puzzles had associated "tips" threads on the forum, released at the same time as the puzzles. These tips suggested ways of thinking about the puzzle, problem decompositions to try, and outline algorithms which students could develop.

### 3.3 Rewards

The event did not formally contribute to the students' qualification in any way, and participation was not enforced. We did not offer any type of leaderboard or reward for early submission of correct answers. However, to encourage continued participation, all students who successfully solved both parts of a problem on the same day were automatically entered into a daily draw for a £20 Amazon voucher. We also held a draw for those who completed at least eight of the ten puzzles by the end of the event, and a draw for respondents in the terminal survey (see section 4.3 below).

## 4 RESULTS

We evaluated this project in three ways: learning analytics of the puzzle submissions, analysis of the forum posts, and an optional terminal survey.

### 4.1 Task submissions

Our learning analytics allows us to count both students to *examined* a task and those who *submitted a solution* to a task.

The event was advertised to around 2500 undergraduate students. 325 engaged with the introductory task and 143 students participated in the main ten days of the Summer of Code. 80 different students made a total of 595 submissions (537 submissions if multiple submissions to the same task are ignored). An additional 165 students accessed only the introduction task; 98 students submitted a solution to this task and 174 students only examined it. In the remainder of this section, we will look only at the ten tasks in the core of the Summer of Code event.

The number of submissions per day declined over the ten days of the activity (Figure 2). Interestingly, there are an appreciable number of participants, termed "lurkers", who looked at each day's puzzle but did not submit a solution.

Students tended to stay the course for the Summer of Code. Of the 80 students who submitted at least one solution, 26 submitted

**Figure 2: Submitters and lurkers per task**

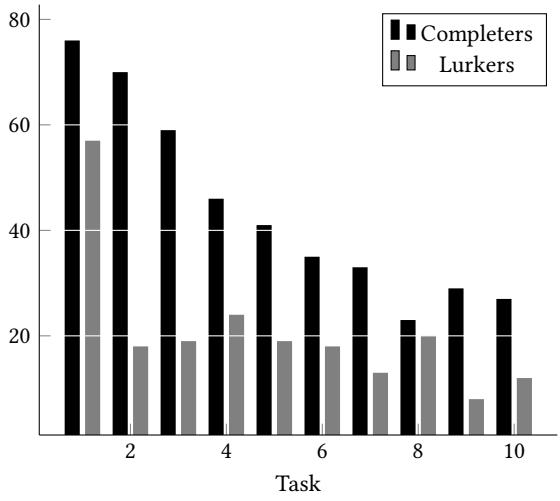
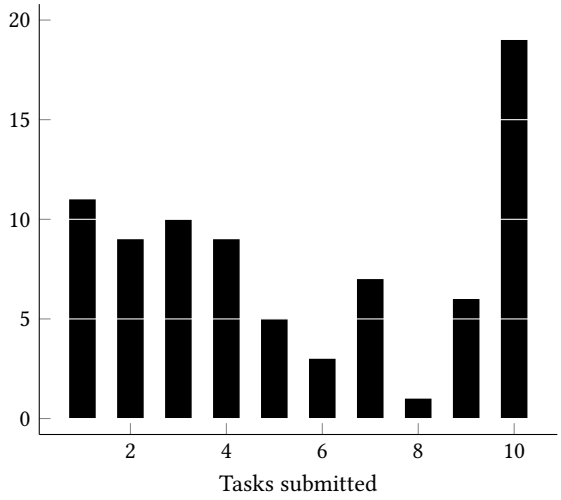



**Figure 3: Number of students who submitted exactly *n* solutions**




solutions on at least eight of the ten days and 19 submitted a solution for each day (Figure 3).

98% of student submissions on each day's task resulted in the student successfully answering both parts, but students sometimes made multiple attempts (Table 1).

Some students attempted some days but not others. Even if a student did not submit a solution to one task, they often remained engaged with the activity. 47 students submitted a solution to a task after they had examined but not submitted a solution for a previous task, meaning that students were still engaged in the activity even if they skipped a task or two. See Table 2 for a breakdown of numbers of student completing and examining different numbers of tasks.

80% of tasks were completed on the same day as they were started. The median duration of a task-solving attempt was 2 hours 45 minutes, though this is an unreliable measure of time spent on

**Table 1: Number of people making this many attempts per task**

| | Attempts | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Task | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 1 | 60 | 13 | 3 | 0 | 0 | 0 | 0 | 0 |
| 2 | 62 | 5 | 2 | 1 | 0 | 0 | 0 | 0 |
| 3 | 56 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | 31 | 10 | 4 | 0 | 1 | 0 | 0 | 0 |
| 5 | 34 | 4 | 1 | 1 | 0 | 0 | 1 | 0 |
| 6 | 18 | 8 | 2 | 2 | 2 | 2 | 1 | 0 |
| 7 | 31 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 8 | 16 | 3 | 1 | 3 | 0 | 0 | 0 | 0 |
| 9 | 27 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 16 | 7 | 2 | 1 | 0 | 0 | 0 | 1 |

**Table 2: Lurkers vs Completers**

| | Lurked | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Completed | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 0 | 49 | 8 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 5 | 4 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | |
| 2 | 1 | 3 | 1 | 0 | 1 | 0 | 1 | 0 | 2 | | |
| 3 | 3 | 2 | 3 | 0 | 1 | 1 | 0 | 0 | | | |
| 4 | 0 | 3 | 3 | 0 | 1 | 1 | 1 | | | | |
| 5 | 2 | 1 | 1 | 0 | 1 | 0 | | | | | |
| 6 | 1 | 0 | 0 | 1 | 1 | | | | | | |
| 7 | 2 | 1 | 1 | 3 | | | | | | | |
| 8 | 0 | 0 | 1 | | | | | | | | |
| 9 | 0 | 6 | | | | | | | | | |
| 10 | 19 | | | | | | | | | | |

a task, as there were some attempts that tasked up to 13 days. If someone started a task, then moved to some other activity, then came back to the task later, the whole duration from start to finish is counted.

Despite there being no benefit to starting the task early, 17% of task completions were started before 1.00am and 11% were finished before 5.00am.

## 4.2 Forum analysis

We encouraged students to post their solution programs in the task discussion forum and to discuss each others' programs. We encouraged discussion as soon as each task opened. There were 192 solutions posted, in a variety of languages. The most common languages were Python, Java, and Sense (a modification of Scratch used on our introductory programming module), which are the languages used in our taught modules. Only 77 of the posted solutions contained any kind of description of the code, either a description in the forum post or comments in the code itself.

Analysis of the posted code was inconclusive. The range of programming languages used and the range of difficulty of the puzzles obscured any trends in program complexity or programming proficiency during the Summer of Code activity.

**Table 3: Lurkers vs Completers**

| Forum name | Authors | Posts | Threads |
|---|---|---|---|
| Announcements | 8 | 31 | 13 |
| Task discussion | 47 | 554 | 35 |
| Show offs' corner | 22 | 94 | 26 |
| No question too simple | 19 | 66 | 13 |

Table 3 shows the participation in the forums. 58 different people posted, with most engaging in the general "task discussion" forum; 25 people posted only there. 15 people posted in both "Show off's corner" and "No question too simple", with discussion going in both directions: experienced programmers were answering "simple" questions, and less-confident programmers were engaging with threads in the "Show off's corner". Participation per author followed a typical distribution, with most posts produced by a small number of users.

The forums were reasonably interactive. In the "task discussion" forum, 206 of the 554 posts referred to another post in the forum, either answering a question or commenting on a posted program. 32 posts indicated that the student was trying out a new programming language or programming technique, and 18 of those posts also referred to another post in the forums. This was a disappointing number of interactions, but perhaps not surprising. Discussions between students were universally generous and supportive. Some sample exchanges are below:

> X: I used the `deque.rotate` method for rotating both rows and columns. Easier.
> <code removed>
> Y: I like the way you parse the instructions—very easy to read.

---

> X: Saw Y used a dispatch table in one of his solutions and it wasn't something I've done before, so I thought I'd try it for this. I think it turned out well.

There were only 23 student posts that referred to the "tips" threads, but all those references commented on how useful the tips threads were.

## 4.3 Terminal survey

We invited all students to participate in a terminal survey, hosted on SurveyMonkey [13]. Participation was not required and could be anonymous, though everyone who provided an email address was entered into a draw for a £25 Amazon voucher. The survey comprised 16 questions and took about three minutes to complete. 58 students completed the survey. The survey was presented near the end of the Summer of Code period, but only 45% of respondents said they had attempted nine or more tasks in the challenge and 22% said they had attempted 3 or 4 tasks. This indicates that the survey captures the results from a range of students.

When asked whether they liked the Summer of Code, 93% of students rated the activity 3 or higher on a 5-point Likert scale

**Table 4: Change in confidence by self-reported skill**

| | Change in confidence | | | | | |
|---|---|---|---|---|---|---|
| Original ability | Not sure | Much less | Little less | No change | Little more | Much more |
| Novice | | 1 | 3 | | 2 | 1 |
| Reluctant | | 1 | | | | |
| Poor | | | 2 | | 1 | 1 |
| Average | | | 5 | 2 | 9 | |
| Good | 2 | 1 | 1 | 10 | 12 | 1 |
| Professional | | | | 3 | | |

(mean 4.44, variance 0.75) and 93% said they would recommend it to other students (the remaining 7% responded "not sure").

60% of respondents said they engaged with the challenge to improve their programming. 66% said they learnt a new skill during the challenge. None said they were motivated by any prizes on offer.

An interesting finding was how the participants' confidence changed as a result of the Summer of Code (Table 4). Respondents were asked, in separate questions, to describe their self-assessed programming ability and how their confidence in programming changed as a result of the Summer of Code. 46% of respondents reported an increase in their confidence while 24% reported a decrease in confidence. Unsurprisingly, self-rated professional programmers did not change in confidence. Most "good" programmers showed an increase in confidence, while other grades of programmer showed a reasonably equal split between those who felt more and less confident.

Changes in confidence did not correlate with satisfaction with the Summer of Code. Seven respondents gave the Summer of Code a rating of 3 or less when asked if they like it. All seven indicated that they felt less confident after the activity, though the other seven responents who felt less confident gave the overall activity a rating of 4 or 5.

We asked participants what they found the most challenging (Table 5). 53% of respondents said that developing a general algorithm or data representation was the most challenging. Only 14% struggled with the programming language and only 10% found debugging the most difficult. (Of the four "other" responses, two wanted to give multiple responses, one was essentially "developing a general algorithm or representation" and one was a comment about their personal lack of time.) This corroborates the comments made on the forum and the survey freetext responses, where students reported they had difficulty bridging the gap between the problem and programming language structures.

Free-response comments showed that the students enjoyed and valued the opportunity to develop their programming skills; some of those comments are below. Although not explicitly asked in the survey, the free-response comments and forum posts showed that less-proficient students enjoyed the opportunity to engage with more proficient students, even if they did not directly learn specific techniques.

> It was good fun, I learnt some useful tricks from viewing the code others used to answer the same questions, I realised how far my programming had come (and also how far it had to go!)

> It is helpful to see how others code and to share ideas about the best way to go about solving a particular programming problem.

> To be practical at programming, practising is vital. The computing modules haven't got enough practice examples with enough variety.

> It has provided a friendly environment for less-experienced programmers to try out their code, and maybe get advice and encouragement, which has to be a good thing. Many thanks!

> When I started, creating a loop which cycled through a list required a little thinking but now the structure is intuitive to me. Debugging and testing meant a quick glance through what I'd written but these activities meant that I had to actually plan my approaches and analyse everything in much more detail. This challenge REALLY helped me!

**Table 5: Greatest challenges**

| Challenge | Responses |
|---|---|
| Understanding the task | 6 |
| General algorithm or representation | 31 |
| Language syntax features | 8 |
| Finding reference materials | 1 |
| Debugging a running program | 6 |
| None | 2 |
| Other | 4 |

## 5 LESSONS LEARNT

We can draw a number of conclusions from this project. First is that there is an appetite for part-time mature students to engage in this extra-curricular activity, despite the myriad other demands on their time. Most students who engaged for two days completed all ten: there was a low drop-out rate over the course of the event. Students of all ability levels seemed to relish the chance to practise and develop their programming skills. Several students started on the puzzles soon after midnight, and there was a burst of submissions each day between 11.00pm and midnight.

The most significant finding for the teaching of programming is about problem solving. As discussed in section 4.3, students found the most challenging part of the puzzles to be the development of an overall approach to solving the problem. Once they had identified that approach, students we able to implement it in their chosen language, and also reported few problems with debugging programs once written. Faculty-provided worked examples of solving the problems were very well received, especially the "think-aloud" sections where faculty described their thought processes for solving

problems. In common with many other undergraduate curricula, the Open University spends a lot of teaching time on language constructs. It seems this effort has paid off, but perhaps at the expense of more practice of analysing problems and decomposing them into computationally-achievable parts.

The students quickly gelled into a supportive community via the event's forums, with many cases of students asking and answering questions, making helpful comments, and engaging in some social chat. This was despite the fact that there were no reports from the students that any of them had previously met, even online. Faculty monitored the forums for conduct, but no intervention was necessary.

Students' limited time was a frequent comment in both the forums and the terminal survey. Many students reported difficulty with finding time to complete the puzzles, especially at the pace of one puzzle per day. In the survey, 40% of respondents self-reported as spending 2–5 hours on each puzzle and 43% of respondents asked for more time between puzzles. Despite that report, 70%–80% of puzzle completions were made on the same day as the puzzle was released.

There is a danger that reducing the pace of puzzles, and spreading the event over a longer period, will increase the number of students who drop out of the event due to other circumstances. We need to balance these factors when presenting the next event.

The number of students participating in the event was disappointing given the size of the cohort who could have engaged. However, there was no compulsion on students to take part, and the event was deliberately pitched between modules so that students had some free time to participate. In addition, the Open University is extremely sensitive to how much information is pushed to students, and we chose to send only two emails to students inviting them to participate. More advertising of the event in other channels, such as being mentioned during current modules, would likely increase the number of students participating.

## 6 CONCLUSIONS

This first Summer of Code exercise was a successful pilot project. We are preparing a follow-up activity in the same vein, but incorporating changes suggested by this pilot. The main changes are to reduce the complexity of the later problems, and to reduce the pace of the problem presentation. We will place greater emphasis on describing the programming schemas used to address these problems, through more explicit descriptions of schemas in the "tips" threads and more "think-aloud" worked example transcripts and videos from experts.

We will also follow the progress of participants in the Summer of Code as they study other modules, to see if their outcomes are different from their peers; a similar activity, aimed at first-year students, increased their confidence but seemed to have no effect on their module outcomes [18].

This activity has addressed the our three research questions:

(1) There is a substantial desire for more learning opportunities, even when the work is not attached to any form of formal award or credit. Half the students who found the activity challenging, and who felt less confident as a result, still rated the activity highly.
(2) Students relished the opportunity to interact with other students with different programming ability. In particular, less-proficient students were not seemingly disheartened by exposure to solutions presented by more-proficient students.
(3) The findings from this project substantially recapitulate the findings of Lahtinen *et al.* [5], that novice programmers have most difficulty with designing an approach to a problem and constructing a program to carry out that task.

This study indicates that a large number of students want to take control of their learning and will pursue additional opportunities to develop their skills and understanding, despite the time pressure that comes from mature students studying part-time.

## ACKNOWLEDGMENTS

We would like to thank Daniel Gooch for insightful comments and suggestions of drafts of this paper.

## REFERENCES

[1] Mark Guzdial. 2003. A media computation course for non-majors. In *ACM SIGCSE Bulletin*, Vol. 35. ACM, 104–108.
[2] Kate S. Hone and Ghada R. El Said. 2016. Exploring the factors affecting MOOC retention: A survey study. *Computers & Education* 98 (2016), 157 – 168. https://doi.org/10.1016/j.compedu.2016.03.016
[3] C. Hu. 2016. Can Students Design Software?: The Answer Is More Complex Than You Think. In *Proceedings SIGCSE '16*. 199–204.
[4] Jong W. Kim, Frank E.Ritter, and Richard J. Koubek. 2013. An integrated theory for improved skill acquisition and retention in the three stages of learning. *Theoretical Issues in Ergonomics Science* 14, 1 (2013), 22–37. https://doi.org/10.1080/1464536X.2011.573008 arXiv:https://doi.org/10.1080/1464536X.2011.573008
[5] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A study of the difficulties of novice programmers. In *Proceedings, ITiCSE '05*. 14–18.
[6] C. Loftus, L. Thomas, and C. Zander. 2011. Can graduating students design: Revisited. In *Proceedings SIGCSE '11*. 105–110.
[7] K.M. Lui and K.C.C Chan. 2006. Pair programming productivity: Novice–novice vs. expert–expert. *Int. J. Human-Computer Studies* 64 (2006), 915–925.
[8] Jerry Mead, Simon Gray, John Hamer, Richard James, Juha Sorva, Caroline St. Clair, and Lynda Thomas. 2006. A Cognitive Approach to Identifying Measurable Milestones for Programming Skill Acquisition. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR '06)*. ACM, New York, NY, USA, 182–194. https://doi.org/10.1145/1189215.1189185
[9] Abhiram G. Ranade. 2016. Introductory Programming: Let Us Cut through the Clutter!. In *Proceedings, ITiCSE '16*. 278–283. https://doi.org/10.1145/2899415.2899430
[10] F. E. Ritter and L. J. Schooler. 2001. The learning curve. 13 (2001), 8602–8605.
[11] Neil Smith. 2017. Summer of Code. (2017). Retrieved 22 January, 2018 from https://github.com/NeilNjae/ou-summer-of-code-2017
[12] E. Soloway. 1986. Learning to program = learning to construct mechanisms and explanations. *Commun. ACM* 29, 9 (1986), 850–âĂŞ858.
[13] SurveyMonkey. 2017. Summer of Code survey. (2017). Retrieved 22 January, 2018 from https://www.surveymonkey.com
[14] Open University. 2016. Facts and Figures 2015–16. (2016). https://www.open.ac.uk/about/main/sites/www.open.ac.uk.about.main/files/files/uk_fact_figures_1516_pdf(1).pdf
[15] Jeroen J.G. van Merrienboer, J.J.G. van Merrienboer, and Fred G.W.C. Paas. 1990. Automation and schema acquisition in learning elementary computer programming : implications for the design of practice. *Computers in human behavior* 6, 3 (1990), 273–289. https://doi.org/10.1016/0747-5632(90)90023-A
[16] Eric Wastl. 2017. Advent of Code. (2017). Retrieved 22 January, 2018 from https://adventofcode.com
[17] L.E. Winslow. 1996. Programming pedagogy—a psychological overview. *ACM SIGCSE Bulletin* 28, 3 (1996), 17–22.
[18] John Woodthorpe. 2017. Personal communication. (2017).